APPLICATION FOR UNITED STATES LETTERS PATENT

INVENTOR: Daniel P. Baumberger
Cornelius, Oregon

TITLE: Efficient Virtual Machine Communication Via Virtual Machine Queues

ATTORNEYS/ AGENTS: Venable, LLP
Box 34385
Washington, DC 20043-9998
Telephone: (202) 344-4000
Facsimile: (202) 344-8300

ATTORNEY DOCKET NO.: 42339-192058

Efficient Virtual Machine Communication Via Virtual Machine Queues

Background of the Invention

Field of the Invention

[0001] Embodiments of the present invention relate generally to a system and method for efficient communication between and among virtual machines, and more particularly to the use of virtual machine queues to communicate between and among virtual machines.

Related Art

[0002] In a computer environment, multiple virtual machines (VMs) can run on one platform at the same time. These virtual machines may contain standard operating systems, such as, *e.g.*, Microsoft Windows® or Linux, or a much smaller operating system. When multiple VMs are running on a platform at a given time, the need may arise for the VMs to communicate with each other.

[0003] Inter-virtual machine (VM) communication currently consists of two different software-based techniques. First, communication via a network is the most common technique because most VMs have network adapters. Second, shared memory is used to map a block of memory into each of the VMs' address spaces that need to communicate. Access to this memory is then synchronized via a software-based semaphore bit or some other synchronization mechanism.

[0004] Each of the above-described techniques has limitations. The networking technique requires full networking stacks in each of the VMs, which creates large overhead. Additionally, both VMs must be running compatible network protocols and be able to discover each other to establish communication. Shared memory is limited by the size of the shared memory block. In other words, it does not work for communication that needs to move variable-sized blocks of data between VMs. Further, the synchronization model is also difficult because each VM needs to have software running to understand the synchronization model.

Brief Description of the Drawings

[0005] Various exemplary features and advantages of embodiments of the invention will be apparent from the following, more particular description of exemplary embodiments of the present invention, as illustrated in the accompanying drawings wherein like reference numbers generally indicate identical, functionally similar, and/or structurally similar elements.

[0006] Figure 1 depicts an exemplary embodiment of a computer system according to an exemplary embodiment of the invention;

[0007] Figure 2 depicts an exemplary arrangement of computer memory in an exemplary embodiment of the present invention;

[0008] Figure 3 depicts an exemplary arrangement of computer memory in an exemplary embodiment of the present invention;

[0009] Figure 4 depicts an exemplary embodiment of a computer system according to an exemplary embodiment of the invention;

[00010] Figure 5 depicts an exemplary flow diagram for illustrating an exemplary method according to the present invention; and

[00011] Figure 6 depicts an exemplary flow diagram for illustrating an exemplary method according to the present invention.

Detailed Description of Exemplary Embodiments of the Present Invention

[00012] Exemplary embodiments of the invention are discussed in detail below. While specific exemplary embodiments are discussed, it should be understood that this is done for illustration purposes only. A person skilled in the relevant art will recognize that other components and configurations can be used without parting from the spirit and scope of the invention.

[00013] An exemplary embodiment of the present invention includes a system, method, and computer program product for highly efficient, variable length data sharing between virtual machines (VMs). Through the use of virtual machine (VM) queues, an exemplary embodiment of the invention provides an instruction-

level mechanism to transfer pages and/or messages between VMs. An exemplary embodiment of the invention further allows variable-sized blocks of data to be moved between VMs in 4KB or 4MB page granularity, for example. In such an embodiment, the central processing unit (CPU) provides synchronization of the VM queues to ensure uniform and secure synchronization.

[00014] Referring now to the drawings, Figure 1 depicts an exemplary embodiment of a computer system 100 according to an exemplary embodiment the present invention. Computer system 100 can have VMs 101a, 101b, a VM monitor 103, VM control structures 102a, 102b and VM queues 104a, 104b. VMs 101a, 101b can contain standard operating systems such as, *e.g.*, Windows® or Linux, or a much smaller operating system. In an exemplary embodiment of the invention, VMs 101a, 101b can be an abstraction of a processor (not shown) for enforcing memory restrictions, input/output restrictions, register restrictions, and the like. Each VM 101a, 101b can define an address space within which to operate. VMs 101a, 101b can also run software within the VM that is limited by the size and other restrictions placed on the VM, for example.

[00015] VM monitor 103 can be software, for example, that can create and manage VMs, handle access to virtual devices, and facilitate sharing and isolation between the VMs.

[00016] As shown in Figure 1, each VM control structure 102a, 102b is associated with a respective VM 101a, 101b. In an exemplary embodiment of the invention, the VM control structures 102a, 102b describe the associated VM in terms of access rights to memory regions and what types of accesses, such as, *e.g.*, memory, input/output, or the like, can cause a VM to exit into the VM monitor 103.

[00017] Each VM control structure 102a, 102b can have an associated VM queue. As shown in Figure 1, VM control structure 102a can be associated with VM queue 104a and VM control structure 102b can be associated with VM queue 104b. VM queues 104a, 104b are mapped into the address space of the VM and referenced by the software of an associated VM.

[00018] During operation of computer system 100, VM queue references are always relative to the currently operating VM because the processor (not shown) only knows about the VM control structure associated with the currently operating VM and the VM monitor. For example, when VM 101a is running, the processor can only access pages or messages in VM queue 104a because the processor only knows about VM control structure 102a and VM monitor 103. If a need arises for the processor to access items in VM queue 104b, VM monitor 103 must cause a VM exit and then give the processor access to VM queue 104b. For purposes of this application, a VM exit can be defined as any event that causes the processor to stop execution of the current VM and return control to the VM monitor including, but not limited to, such events as faults, exceptions, interrupts, memory accesses, IO accesses, control register access, special virtualization instructions, etc. A "natural" VM exit can be any event that causes a VM exit that is not an explicit VM exit request by the software running in the VM, i.e. a VM exit that is not caused by an instruction intended specifically to cause a VM exit.

[00019] Computer system 100 can provide an exemplary method to allow VM 101a to communicate with VM 101b via VM queues 104a, 104b, for example. Figure 2 depicts an exemplary embodiment of an initial memory allocation between to VMs using a simple memory mapping technique as would be recognized by a person having ordinary skill in the art. Figure 2 shows guest memory 201a of VM 101a, guest memory 201b of VM 101b, page table 202, and physical host memory 203. In an exemplary embodiment of the invention, guest memory 201a, 201b can be the virtual address space within which a VM can operate. As shown in Figure 2, VM 101a receives 128MB, for example, of guest memory, and the VM monitor 103 (not shown in Figure 2) establishes the range of guest memory 201a from 0-127MB. Similarly, VM 101b receives 128MB, for example, of guest memory, and the VM monitor 103 establishes the range of guest memory 201a from 0-127MB. In this example, the addresses for guest memory 201a match those of physical host memory 203. The addresses for guest memory 201b are offset by 128MB.

[00020] Page table 202 can automatically map guest memories 201a, 201b into host physical memory 203. Each VM 101a, 101b, however, can only access the respective 128MB, for example, of guest memory 201a, 201b that the VM 101a, 101b was assigned. Further, each VM 101a, 101b interprets the range of guest memory 201a, 201b to begin with zero.

[00021] Figure 3 depicts an exemplary method for remapping a page in address space 201b of VM 101b into the address space 201a of VM 101a. As an example, the page inside address 204b of guest memory 201b of VM 101b is initially associated with address 205 of host physical memory 203 (as indicated by the broken lines). When the page table 202 is updated by the processor (not shown), the page inside address 204a of guest memory 201a of VM 101a now points to address 205 of physical host memory 203. By updating page table 202, the page associated with VM 101b is now associated with VM 101a.

[00022] Figure 4 depicts an exemplary computer system 400 for communicating between two VMs using VM queues. Computer system 400 can include originating VM 401a, destination VM 401b, originating VM control structure 402a, destination VM control structure 402b, VM monitor 403, originating VM queue 404a, and destination VM queue 404b. Originating VM 401a is associated with originating VM control structure 402a and originating VM queue 404a through VM monitor 403. Similarly, destination VM 401b is associated with destination VM control structure 402b and destination VM queue 404b through VM monitor 403.

[00023] VM queues 404a, 404b can be used to update a page table, and thus allow two or more VMs to communicate. For example, if originating VM 401a desires to communicate with destination VM 401b, originating VM 401a can place a message, such as, *e.g.*, a 4KB or 4MB page, into VM queue 404b of destination VM 401b. In an exemplary embodiment of the invention, the amount of data being transferred during successive communications between VMs can vary and does not have to be constant.

[00024] Figure 5 depicts flow diagram 500 which illustrates an exemplary method according to the present invention for placing a message into destination VM queue 404b from originating VM 401a, for example. It will be understood by a person having ordinary skill in the art that the exemplary method can be reversed to allow destination VM 401b to place a message into originating VM queue 404a.

[00025] In step 501, originating VM 401a can determine whether the write instruction that ultimately places the message into the destination VM queue requires an immediate VM exit. If the write instruction requires an immediate VM exit, flow diagram 500 can proceed to step 502. In step 502, the write instruction can cause an immediate VM exit so that VM monitor 403 can process the delivery of the message. If the write instruction does not require an immediate VM exit, flow diagram 500 can proceed to step 503.

[00026] In step 503, the write instruction can include a so-called "lazy write policy." In other words, the write instruction does not cause an immediate VM exit from originating VM 401a, but places the message into originating VM control structure 402a to be delivered when originating VM 401a causes a "natural" VM exit of originating VM 401a during step 504.

[00027] During step 505, once originating VM 401a exits, for example, via an immediate VM exit or a "natural" VM exit, VM monitor 403 can deliver the message to destination VM 401b by mapping the page into the address space (not shown in Figure 4) of destination VM 401b and placing the message into destination VM queue 404b. Once the page has been placed in destination VM queue in step 505, flow diagram 500 can proceed to step 506. In step 506, VM monitor 403 can resume and/or schedule destination VM 401b and execute an instruction to read a message from VM queue 404b. Once step 506 is completed, the next message waiting in destination VM queue 404b can be dequeued during step 507. Once the message had been dequeued in step 507, flow diagram 500 can proceed to step 508. During step 508, destination VM 401b can process the

message. As will be understood by a person having ordinary skill in the art, destination VM 401b can use software to process the message.

[00028] In an exemplary embodiment of the invention, an instruction can be provided for determining the length of a VM queue. Determining the length of a VM queue can advantageously avoid VM exits when the VM control structure is accessing an empty queue, for example. If no message is in the VM queue (associated with the currently executing VM), a VM exit will occur to allow the VM monitor to schedule another VM to execute.

[00029] Figure 6 depicts flow diagram 600, which illustrates an exemplary embodiment of the dequeueing step of an exemplary method according to the present invention. During step 601, depending upon the software running in the currently executing VM, the VM control structure of the currently executing VM can determine the length, *i.e.*, the number of messages, waiting in the queue. If the queue is empty, flow diagram can proceed to step 602. During step 602, a VM exit can occur so that the VM monitor can schedule another VM to execute during step 605 or, alternatively, the software running in the VM can choose to continue performing other tasks. If the VM queue is not empty, flow diagram can proceed to step 603. During step 603, the address can be stored into a local variable associated with the currently executing VM.

[00030] In an exemplary embodiment of the invention, an instruction can be provided for conveying identification information from one VM to another VM. In this exemplary embodiment, a software entity that knows about the establishment of the VMs can be used to write to the queues of the other VMs. For example, software running as part of the VM monitor or an interface to the VM monitor can obtain identification information to be conveyed to other VMs. After the identification information of one VM has been conveyed to another VM, the software or interface can then establish a protocol using the VM queues. For example, in an exemplary embodiment of the invention, if multiple VMs are used to accomplish a single task, such as, *e.g.*, play an audio file, each of the multiple VMs can be used to accomplish an individual task, such as, *e.g.*, read the file,

decode the file, and write to the audio device. In such an example, as would be understood by a person having ordinary skill in the art, each of the VMs would need to know information necessary to communicate with the VM that accomplishes the next task. For example, the "read" VM would need to know information about the "decode" VM so that the "read" VM is able to pass the file that is read to the "decode" VM using, for example, the exemplary method described above.

[00031] When the "read" VM passes the file to the "decode" VM, software running in association with the "read" and "decode" VMs can exchange VM identification information so that the VM monitor can then establish their protocol via the VM queue mechanism.

[00032] The embodiments illustrated and discussed in this specification are intended only to teach those skilled in the art the best way known to the inventors to make and use the invention. Nothing in this specification should be considered as limiting the scope of the present invention. All examples presented are representative and non-limiting. The above-described embodiments of the invention may be modified or varied, without departing from the invention, as appreciated by those skilled in the art in light of the above teachings. It is therefore to be understood that the invention may be practiced otherwise than as specifically described.